POLITECNICO
MILANO 1863

Integration Test Plan Document

version 1.0

15th January 2017

Authors:
Davide Moneta (808686)
Federico Oldani (806337)
Luca Oppedisano (878301)

Index

1 Introduction

1.1 Purpose

The Integration Testing Plan Document for the PowerEnJoy system has the purpose to show and explain the organization of the integration testing activities for all components. We will discuss the testing strategy, order, content, requirements and the reasons for our choices.

1.2 Scope

We will design a software to manage the functionalities of PowerEnJoy: a new car sharing service. PowerEnJoy offers exclusively electric cars that allow the users to move around respecting the environment.
Using this platform customers are able to find and reserve one of our cars within a certain distance from their current location or a specified address. When they reach the chosen car, they can unlock it and drive it.
Users are charged with a per-minute fee, but the company incentivizes virtuous behaviors by offering a discount at the end of a trip if certain conditions are met.

1.3 Definitions and Abbreviations

Visitor: a generic person who access the website or the app without being authenticated
User: equivalent to registered user, a visitor who already performed the registration and has now access to the system
Car: with this term we always refer to one of our cars
Car tag: a term to label the status of a car, it can be "available" or "reserved" or "in use"
Available car: a car that is currently not in use nor reserved by anyone and has at least 30% of battery level
Reservation: it can be requested by a user for an available car, this gives the user the possibility to unlock the reserved car within a time span of 1 hour from the reservation time, after this time the reservation expires
Reserved car: a car that can be unlocked only by the one who reserved it
Ignition code: this code is sent to the user when a reservation is successful, user will have to type this code on the OBS lock screen in order to ignite the engine
Locking a car: after the passengers are all out of the car and the door are all closed the system counts 40 seconds and then locks the car. There are two types of locking: if the user wants to keep the car on pause, the car is locked and it remains on pause for that user, otherwise the car is locked and made available to other users. System locks the car only if the car is inside the Safe Area

Pause: when the user turns off the engine the systems asks him on the OBS if he wants to put the car on pause, if the user taps "YES" the pause state is applied: the user keeps paying the per-minute fee until he unlocks it again
Ride or Trip: it starts when the engine is ignited or 2 minutes after the car is unlocked, and ends when the car is locked
Temporary bill: keeps track of the duration of the trip or the pause to calculate the amount of money to pay
Safe area: we consider a safe area the set of every possible legal parking spot within The Area, so all the places where it is possible to park the car without being fined
SPA: these are Special Parking Areas where cars can be parked and also recharged through a special plug
The System: the personification of the software, which manages all the functionalities of PowerEnJoy
MSM: Money Saving Mode, this can be enabled in the car after selecting a destination, this ensures to find parking solution to get a discount, optimizing the cost of the trip; the solution takes also into account the position of cars in The Area to ensure a uniform distribution
OBS: On Board Screen, it's the display inside the car, where the user can visualize informations about the car and the trip

1.4 List of Reference Documents

- The project documentation: RASD v1.1 & DD
- Integration Test Plan Example.pdf
- Integration testing example document.pdf
- Project assignment

2 Integration Strategy

2.1 Entry Criteria

Before starting with the integration tests, some criterias have to be met:
- RASD and DD delivered, to have a full picture of the system
- At least 50% Unit Test coverage of User and Staff application
- At least 80% Unit Test coverage of PowerEnJoy subsystem
- At least 90% Unit Test coverage of OnBoard subsystem

User and Staff application Unit Test will continue up to 90% coverage during the integration tests.

2.2 Elements to be Integrated

The philosophy behind our architecture is that each component must have limited responsibility, so the interactions between components are fundamental and must be tested enough to have a reasonable guarantee of proper functioning. The reliability of our system strongly depends on the coverage of the unit tests.

We will refer to couples of component by naming the interface between them, so when we say that an interface will be tested we mean that the components linked by that interface will be tested through that interface.

One interface in our system plays a trivial role, because only a basic knowledge of query language is required to make it work, but it is fundamental so it will also be tested, this interface is:

- DatabaseAccess

The following interfaces operate with components developed by third-party developers, so it's necessary to test how they integrate with our system:

- Payment
- CoordinateInterface
- MapInterface
- SensorInterface

We will also test all the remaining interfaces, to make sure that the communication between the internal components is not causing unexpected behaviors.

2.3 Integration Testing Strategy

We are going to use a bottom-up approach for the integration tests. With this approach we can base our testing on real components and not on stubbed ones and the integration will follow the implementation order, so that a certain interface will be tested when all the required components are ready.

PowerEnjoy system is composed by three main blocks:

- User and Staff applications
- On Board System
- Server subsystem

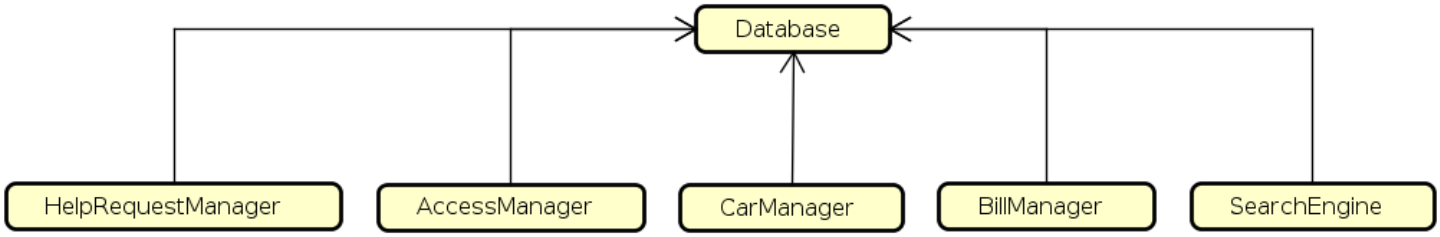
We are going to integrate all the components within the Server subsystem, the two components of the On Board System and then the two blocks together. Finally we will concentrate on the integration with User and Staff application.

2.4 Sequence of Component/Function Integration

In this paragraph we are going to present all the steps of the testing phase. The arrows mean that there is a function call going from the tail to the head.

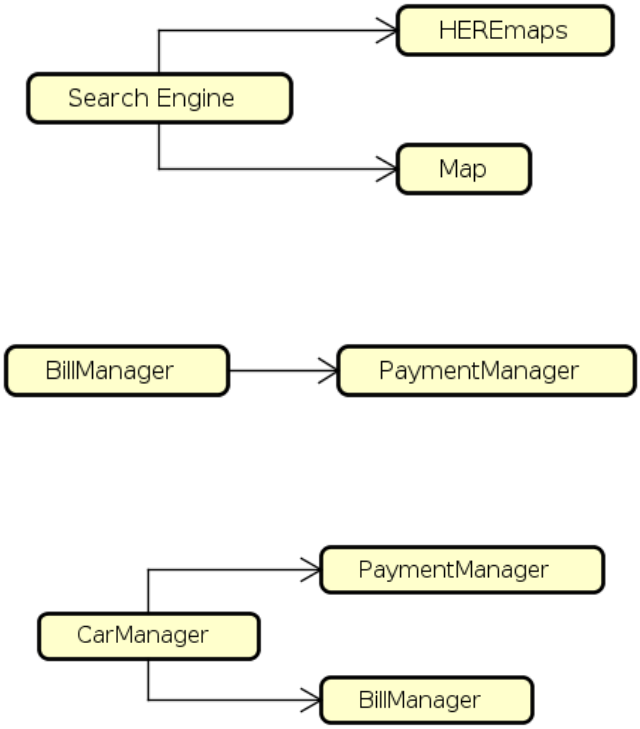
2.4.1 Step 1

The first integration tests required are between the main components of the Server subsystem and the cloud Database.



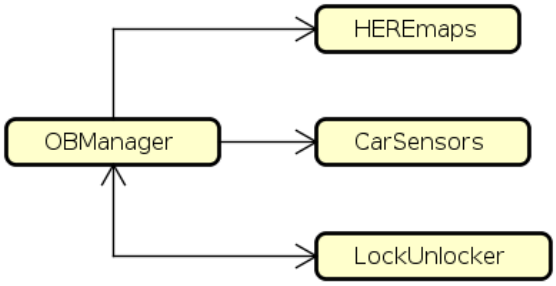
2.4.2 Step 2

Now that the Database is integrated we can proceed with the following pair of components related to the Server:



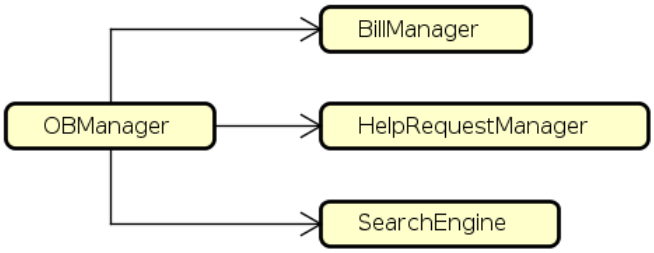
2.4.3 Step 3

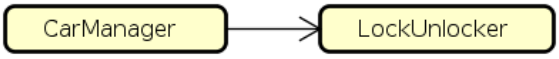
This step is dedicated to integrate the internal components of the On Board System and the third-party components



2.4.4 Step 4

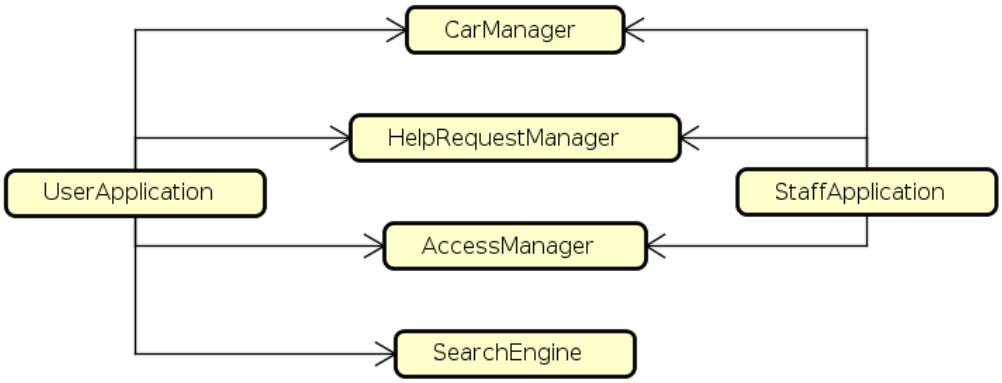
Now that the Server subsystem and the On Board System are internally integrated we can integrate them with each other





2.4.5 Step 5

The last step is to integrate the application with the Server



3 Individual Steps and Test Description

3.1 Step 1

3.1.1 DatabaseForAccessManager (AccessManager, Database)

createNewUser(userData)	
Input	**Effect**
Empty fields	Query not sent, IllegalArgumentException
Driving licence or payment method not formally correct	Query not sent, IllegalArgumentException
Mail or driving licence already registered	No insertion of new tuple, return false
All fields non empty and formally correct, mail not already in use	Creation of a new tuple for the new user, return true

checkCredentials(mail, password)	
Input	**Effect**
Null fields	Query not sent, IllegalArgumentException
Mail not formally correct	Query not sent, IllegalArgumentException
Mail and password not matching	Access denied, returns false
Mail and password matching	Access granted, returns true

3.1.2 DatabaseForCM (CarManager, Database)

getMyReservedCar(userID)	
Input	**Effect**
Invalid ID	IllegalArgumentException
Valid user ID	Returns information about the reserved car

changeCarState(state)	
Input	**Effect**
Valid Tag	Set the state of the car in the database to the one specified
Same state as before	Nothing change, throws NotAllowedException
Null Tag	IllegalArgumentException

3.1.3 DatabaseForBill (BillManager, Database)

getBillHistory(user)	
Input	**Effect**
null or invalid parameter	IllegalArgumentException
valid user(id)	returns the user's bill history

registerBill(bill)	
Input	**Effect**
NULL value	IllegalArgumentException No insertion of new tuple
Correct Bill	Insertion of the Bill in the Database

3.1.4 DatabaseForSearchEngine (SearchEngine, Database)

getCars(coord, radius)	
Input	**Effect**
Inconsistent coordinate input	IllegalArgumentException
Negative or zero radius	IllegalArgumentException
Area with no available car inside	Return an empty list
Area with some available cars inside	Return the list of available Cars in that area

3.1.5 DatabaseForHelpRequest (HelpRequestManager, Database)

logHelpRequest(helpRequest)	
Input	**Effect**
Null help request	IllegalArgumentException
Valid help request	Request saved in Database

3.2 Step 2

3.2.1 MapInterface (SearchEngine, HEREmaps)

addressToCoordinates(address)	
Input	**Effect**
Empty string	IllegalArgumentException
Ambiguous or generic address with more than one match, not existing address	No coordinates returned
Unambiguous address	Return the only possible coordinates

getMap(location, radius)	
Input	**Effect**
Null parameters or negative radius	IllegalArgumentException
Valid area	Return map of that area

markCoordinatesOnMap(map, cars<>)	
Input	**Effect**
Valid map and non empty list of cars	Return a map with car's position marked
Invalid map or empty list of cars	IllegalArgumentException

3.2.2 CoordinateInterface (SearchEngine, Map)

isInSPA(coordinate)	
Input	**Effect**
Null or not formally correct coordinate	IllegalArgumentException
Coordinates inside a SPA	Return true
Coordinates out of a SPA	Return false

isInSafe(coordinate)	
Input	**Effect**
Null or not formally correct coordinate	IllegalArgumentException
Coordinates inside a SafeArea	Return true
Coordinates out of a SafeArea	Return false

3.2.3 Payment (BillManager, PaymentManager)

pay(paymentData)	
Input	**Effect**
Null parameter or invalid parameter	IllegalArgumentException Transaction aborted
Valid payment data	Transaction carried out

3.2.4 BillForCM (CarManager using BillManager)

createBill(userID)	
Input	**Effect**
NULL or invalid user ID	IllegalArgumentException
Valid user ID	A new bill is created for the user

getBillInfo(userID)	
Input	**Effect**
Invalid user ID or valid user with no active Bill	IllegalArgumentException
Valid user ID	Returns the information about the current Bill of the user

3.2.5 Payment (CarManager, PaymentManager)

pay(paymentData)	
Input	**Effect**
Null parameter or invalid parameter	IllegalArgumentException Transaction aborted
Valid payment data	Transaction carried out

3.3 Step 3

3.3.1 MapInterface (OBManager, HEREmaps)

addressToCoordinates(address)	
Input	**Effect**
Empty string	IllegalArgumentException
Ambiguous or generic address with more than one match	No coordinates returned
Unambiguous address	Return the only possible coordinates

3.3.2 SensorInterface (OBManager using CarSensors)

getNumberOfPassengers()	
Input	**Effect**
---	Return the correct number of passengers detected by sensors

areDoorsClosed()	
Input	**Effect**
---	Return false if at least one door is open, true otherwise

3.3.3 LockerForOBM (OBManager, LockUnlocker)

lockDoors()	
Input	**Effect**
---	Return true if operation successful, false otherwise

3.3.4 TurnOn (LockUnlocker, OBManager)

wakeUp(ignitionCode)	
Input	**Effect**
Well formed ignitionCode	OBManager is woken up from sleep mode and returns true
Null or non formally correct ignitionCode	OBManager is woken up from sleep mode, returns false and it goes back in standby

3.4 Step 4

3.4.1 BillForOBM (OBManager, BillManager)

closeBill(pause)	
Input	**Effect**
not boolean value	IllegalArgumentException
boolean value *true*	Bill stops counting and charges the user but the bill persists in memory
boolean value *false*	Bill stops counting and charges the user

3.4.2 SearchForOBM (OBManager, SearchEngine)

searchSPA(address)	
Input	**Effect**
Empty string	IllegalArgumentException No SPA returned
Ambiguous or generic address with more than one match	No coordinates returned
Unambiguous address	Return the coordinates nearest SPA to the address

3.4.3 HelpRequestForUser (OBManager, HelpRequestManager)

enqueue(request)	
Input	**Effect**
NULL or not HelpRequest object	IllegalArgumentException
HelpRequest is already enqueued	The methods doesn't enqueue any request
HelpRequest is valid	The helprequest is enqueued

3.4.4 UnlockerForCM (CarManager, LockUnlocker)

unlockDoors(ignitionCode)	
Input	**Effect**
Well formed ignitionCode	Return true if operation successful, false otherwise
Formally incorrect or Null ignitionCode	IllegalArgumentException

3.5 Step 5

3.5.1 Login (UserApplication, AccessManager)

login(loginData)	
Input	**Effect**
Email doesn't exist or null value	Returns IllegalArguementValueException
Email and password doesn't match	Returns InvalidCredentialError
Email and password match	Returns true

register(clientData)	
Input	**Effect**
Null parameter or invalid clientData	Returns false
ClientData already exists	Returns ExistingUserException
ClientData are valid	Returns true

3.5.2 SearchForUser (UserApplication, SearchEngine)

getCarByAddress(address, radius)	
Input	**Effect**
Inconsistent or empty address input	IllegalArgumentException
Negative or zero radius	IllegalArgumentException
Area with no available car inside	Return an empty list
Area with some available cars inside	Return the list of available Cars in that area

getCarByCoordinates(coordinates, radius)	
Input	**Effect**
Inconsistent or empty coordinate input	IllegalArgumentException
Negative or zero radius	IllegalArgumentException
Area with no available car inside	Return an empty list
Area with some available cars inside	Return the list of available Cars in that area

3.5.3 CarController (UserApplication, CarManager)

getInfoCar(carID)	
Input	**Effect**
Valid CarID passed as parameter	Returns details of the car
Null or invalid parameter	IllegalArgumentException

unlockRequest(carID,position)	
Input	**Effect**
Well formed CarID and GPS position within 10m from the car	Execute the request and return the same of the *unlockDoors* method
Well formed CarID and GPS position farther than 10m from the car	Return false
Null or non formally correct parameters	IllegalArgumentException

reserve(carID)	
Input	**Effect**
CarID is null or refers to a not available car	IllegalArgumentException
CarID is a valid ID	Return true, car is tagged as *Reserved*

3.5.4 HelpRequestForUser (UserApplication, HelpRequestManager)

enqueue(request)	
Input	**Effect**
NULL or not HelpRequest object	IllegalArgumentException
HelpRequest is already enqueued	The methods doesn't enqueue any request
HelpRequest is valid	The helpRequest is enqueued

3.5.5 Login (StaffApplication, AccessManager)

login(loginData)	
Input	**Effect**
Email doesn't exist or null value	Returns IllegalArguementValueException
Email and password doesn't match	Returns InvalidCredentialError
Email and password match	Returns true

3.5.6 CarController (StaffApplication, carManager)

getInfoCar(carID)	
Input	**Effect**
Valid CarID passed as parameter	Returns details of the car
Null or invalid parameter	IllegalArgumentException

3.5.7 HelpRequestForStaff (StaffApplication, HelpRequestManager)

extractFirstRequest()	
Input	**Effect**
---	Returns the first helpRequest in the queue

4 Tools and Test Equipment Required

4.1 Tools

For the phase of testing of the different components used in the PowerEnjoy system we are going to use some of the most reliable automated testing tools.

In particular we will use two tools for the logic components that are running in the JEE environment: **Arquillian** integration testing and **JUnit** frameworks.

We will use the first one to execute tests on a Java container in order to check the interaction between a component and the environment surrounding it.

JUnit framework is primarily used to make unit testing, but it can be used to check that the interaction between components is working as expected. Specifically we are going to use it in order to: check if the right exception is thrown when an invalid parameter is passed, check what object is returned by methods and other problems that can come up during the integration of different components.

In order to test the user interfaces and ensure that they are suitable for the vast majority of potential users of the PowerEnjoy system (for RAM, storage dimensions and CPU performances needed) we're going to use different tools for every device.

For the Smartphone app we have to divide the testing part depending on the OS installed:

- Android have a list of tools, accessible from the settings, that we use to test the memory occupied in storage by our app, the portion of RAM used to run it and the CPU usage. These are allocation tracker, Memory Monitor and CPU monitor.
- On iOS we use the CPU Activity log, Activity monitor and Manage Storage (that are tools already present on the devices) to check CPU, RAM and storage.
- For Windows Phone we use the Windows Performance Analyzer tool that is included in the toolkit: Windows Phone Performance Analysis.

For the Web App we use the Task Manager on Google Chrome, on Mozilla Firefox, on Safari, on Microsoft Edge and on Opera to monitor the performance of the system when using our web app.

4.2 Test Equipment

The test environment for the App will make use of a variety of smartphones in order to be sure to cover the widest range of different configurations. To do so we will use at least an android smartphone, an android tablet, an iPhone, an iPad, a Windows Phone smartphone, a WP tablet and a PC with the 5 most common browser written above.

The server and the database used after the deployment are the same used to test The System.

In order to test the hardware components managed by CarSensor and LockUnlocker, we will use a real PowerEnJoy car.

5 Program Stubs and Test Data Requirement

5.1 Program Stubs and Drivers

Due to the limited responsibility that we gave to each component and bottom-up approach we need to have a significant number of Drivers to simulate methods invocations on the components. The necessary drivers are described below.

- **HelpRequestManager Driver**: this module performs the invocation of the methods exposed by HelpRequestManager to test their interaction with the DB component
- **AccessManager Driver**: this module will invoke the AccessManager to test its interaction with the DB component
- **CarManager Driver**: this module will invoke the CarManager to test its interaction with the DB, PaymentManager, BillManager and LockUnlocker components
- **BillManager Driver**: this module will invoke the BillManager to test its interaction with the DB and PaymentManager components
- **SearchEngine Driver**: this module will invoke the SearchEnigne to test its interaction with the DB, HereMaps and Map components
- For the OnBoardSystem we will use a **OBManager Driver** which will invoke the OBManager to test its interaction with the HereMaps, CarSensors, HelpRequestManager and LockUnlocker components.
- **UserApplication Driver**: this module will invoke the UserApplication to test its interaction with the CarManager, HelpRequestManager, SearchEngine and AccessManager components
- **StaffApplication Driver**: this module will invoke the StaffApplication to test its interaction with the CarManager, HelpRequestManager and AccessManager components

5.2 Test Data

In order to properly check every case possible in the tests above we need a set of examples that have to include the following unusual inputs to test every part of The System:

- Null object
- Parameter in the wrong format
- Password too short
- Wrong password
- E-mail already existent in the database (when registering)
- E-mail not existent in the database (when logging in)
- Coordinates of a place inside the SPAs
- Coordinates of a place inside the Safe Area but outside the SPAs
- Coordinates of a place outside the Safe Area
- UserID not existent
- Same car Tag in which the car were before (changing the car state)
- Invalid coordinates
- Invalid radius (radius≤0)
- Area containing no cars
- Generic address with more than one place associated
- Invalid address
- Invalid map
- Empty list of cars
- Invalid userID
- UserID with no active Bill (passed to getBillInfo)
- Invalid paymentData
- Empty string
- HelpRequest already enqueued
- Invalid CarID
- CarID referred to a NOT available car

6 Effort Spent

6.1 Moneta Davide

21/12/2016	1h00’
07/01/2017	1h00’
09/01/2017	1h00’
10/01/2017	2h00’
11/01/2017	2h30’
12/01/2017	5h10’

6.2 Oldani Federico

21/12/2016	1h00’
28/12/2016	2h00’
31/12/2016	1h30’
03/01/2017	1h00’
06/01/2017	1h00’
07/01/2017	3h30’
09/01/2017	3h30’
11/01/2017	3h30’
12/01/2017	3h30’
13/01/2017	0h30’
14/01/2017	0h30’

6.3 Oppedisano Luca

21/12/2016	1h00’
28/12/2016	1h00’
03/01/2017	1h00’
04/01/2017	3h30’
07/01/2017	4h30’
09/01/2017	1h00’
10/01/2017	1h00’
11/01/2017	4h00’
12/01/2017	3h45’
13/01/2017	0h30’
14/01/2017	0h30’